CS 103 Unit 10 Slides

C++ Classes

Mark Redekopp

Object-Oriented Programming

- Model the application/software as a set of objects that interact with each other
- Objects fuse data (i.e. variables) and functions (a.k.a methods) that operate on that data into one item (i.e. object)
  - Like structs but now with associated functions/methods
- Objects become the primary method of encapsulation and abstraction
  - Encapsulation
    - Hiding of data and implementation details (i.e. make software modular)
    - Only expose a well-defined interface to anyone wanting to use our object
  - Abstraction
    - How we decompose the problem and think about our design rather than the actual code

Objects

- Objects contain:
  - Data members
    - Data needed to model the object and track its state/operation (just like structs)
  - Methods/Functions
    - Code that operates on the object, modifies it, etc.
- Example: Deck of cards
  - Data members:
    - Array of 52 entries (one for each card) indicating their ordering
    - Top index
  - Methods/Functions
    - Deck.Shuffle(), Deck.Cut(), Deck.Deal()

C++ Classes

- Classes are the programming construct used to define objects, their data members, and methods/functions
- Similar idea to structs
- Steps:
  - Define the class' data members and function/method prototypes
  - Write the methods
  - Instantiate/Declare object variables and use them by calling their methods
- Terminology:
  - Class = Definition/Blueprint of an object
  - Object = Instance of the class, actual allocation of memory, variable, etc.

```
#include <iostream>
using namespaces std;
class Deck {
 public:
   Deck();    // Constructor
   int deal();
 private:
   int cards[52];
   int top_index;
};

// member function implementation
Deck::Deck()
{
  for(int i=0; i < 52; i++)
    cards[i] = i;
}
int Deck::deal()
{
   return cards[top_index++];
}
// Main application
int main(int argc, char *argv[]) {
  Deck d;
  int hand[5];
  d.shuffle();
  d.cut();
  for(int i=0; i < 5; i++){
    hand[i] = d.deal();
  }
  ...
```

poker.cpp

C++ Classes

- Classes are the programming construct used to define objects, their data members, and methods/functions
- Similar idea to structs
- Steps:
  - Define the class’ data members and function/method prototypes **(usually in a separate header file)**
  - Write the methods **(usually in a separate .cpp file)**
  - Instantiate/Declare object variables and use them by calling their methods
- Terminology:
  - Class = Definition/Blueprint of an object
  - Object = Instance of the class, actual allocation of memory, variable, etc.

deck.h

```
class Deck {
 public:
   Deck();    // Constructor
   ~Deck();   // Destructor
   void shuffle();
   void cut();
   int deal();
 private:
   int cards[52];
   int top_index;
};
```

deck.cpp

```
#include<iostream>
#include “deck.h”


// Code for each prototyped method
```

poker.cpp

```
#include<iostream>
#include “deck.h”

int main(int argc, char *argv[]) {
  Deck d;
  int hand[5];

  d.shuffle();
  d.cut();
  for(int i=0; i < 5; i++){
    hand[i] = d.deal();
  }
}
```

Class Definition

- class *name* { … };
- Each function or data member can be classified as public, private, or protected
  - These classifications support encapsulation by allowing data/method members to be inaccessible to code that is not a part of the class (i.e. only accessible from within a public class method)
  - Ensure that no other programmer writes code that uses or modifies your object in an unintended way
  - **Private**: Can call or access only by methods/functions that are part of that class
  - **Public**: Can call or access by any other code
  - **Protected**: More on this later
- Everything private by default so you must use “public:” to make things visible
- Make the interface public and the guts/inner-workings private

deck.h

```
class Deck {
 public:
   Deck();    // Constructor
   ~Deck();   // Destructor
   void shuffle();
   void cut();
   int deal();
 private:
   int cards[52];
   int top_index;
};
```

deck.cpp

```
#include<iostream>
#include “deck.h”


// Code for each prototyped method
```

poker.cpp

```
#include<iostream>
#include “deck.h”

int main(int argc, char *argv[]) {
  Deck d;
  int hand[5];

  d.shuffle();
  d.cut();

  d.cards[0] = ACE; //won’t compile
  d.top_index = 5;  //won’t compile
}
```

Constructors / Destructors

- **Constructor** is a function of the same name as the class itself
  - It is called automatically when the object is created (either when declared or when allocated via ‘new’)
  - Use to initialize your object (data members) to desired initial state
  - **Returns nothing**
- **Destructor** is a function of the same name as class itself with a ‘~’ in front
  - Called automatically when object goes out of scope (i.e. when it is deallocated by ‘delete’ or when scope completes)
  - Use to free/delete any memory allocated by the object
  - **Returns nothing**
  - **[Note: Currently we do not have occasion to use destructors; we will see reasons later on in the course]**

deck.h

```
class Deck {
 public:
   Deck();    // Constructor
   ~Deck();  // Destructor
   ...
};
```

deck.cpp

```
#include<iostream>
#include “deck.h”


Deck::Deck() {
  top_index = 0;
  for(int i=0; i < 52; i++){
    cards[i] = i;
  }
}

Deck::~Deck() {
}
```

poker.cpp

```
#include<iostream>
#include “deck.h”

int main(int argc, char *argv[]) {
  Deck d;  // Deck() is called
  ...

  return 1;
  // ~Deck() is called since
  // function is done
}
```

Writing Member Functions

- When writing member functions, the compiler somehow needs to know that the function is a member of a particular class and that the function has inherent access to data members (w/o declaring them). Thus we must ‘scope’ our functions
- Include the name of the class followed by ‘::’ just before name of function
- This allows the compiler to check access to private/public variables
  - Without the scope operator [i.e. void shuffle() rather than void Deck::shuffle()] the compiler would think that the function is some outside function (not a member of Deck) and thus generate an error when it tried to access the data members (i.e. cards array and top_index).

deck.h

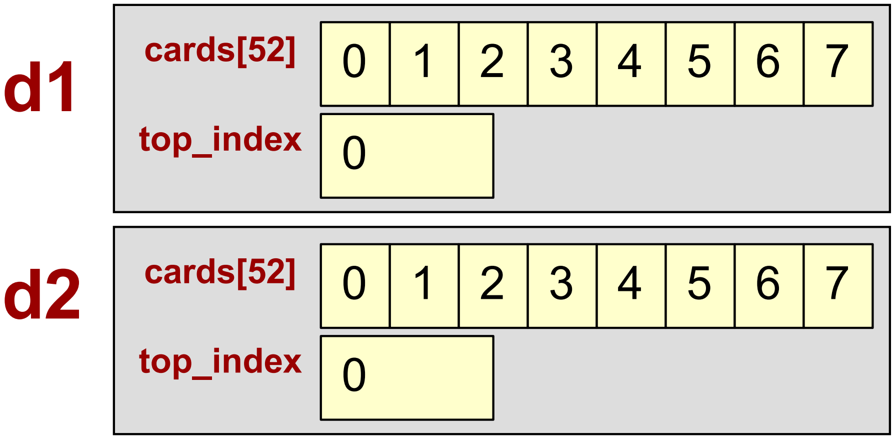
```
class Deck {
 public:
   Deck();    // Constructor
   ~Deck();   // Destructor
   ...
};
```

deck.cpp

```
#include<iostream>
#include “deck.h”

Deck::Deck() {
  top_index = 0;
  for(int i=0; i < 52; i++){
    cards[i] = i;
  }
}

Deck::~Deck()
{
}

void Deck::shuffle()
{
  cut(); //calls cut() for this object
  ...
}

int Deck::deal()
{
  top_index++;
  return cards[top_index-1];
}
```

Calling Member Functions

- Member functions are called by preceding their name with the specific object that it should operate on
- d1.shuffle() indicates the code of shuffle() should be operating implicitly on d1's data member vs. d2 or any other Deck object



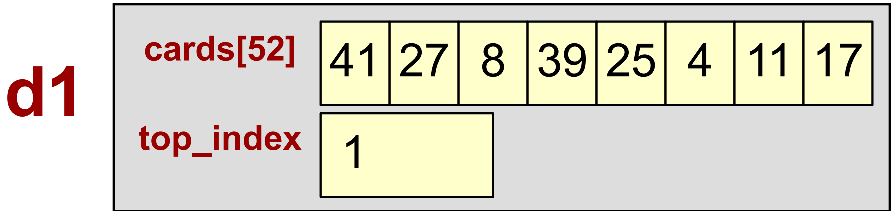
d1

cards[52]	0	1	2	3	4	5	6	7
top_index	0							

d2

cards[52]	0	1	2	3	4	5	6	7
top_index	0							

```
#include<iostream>
#include "deck.h"

int main(int argc, char *argv[]) {
  Deck d1, d2;
  int hand[5];

  d1.shuffle();
  // not Deck.shuffle() or
  // shuffle(d1), etc.

  for(int i=0; i < 5; i++){
    hand[i] = d1.deal();
  }
}
```



d1

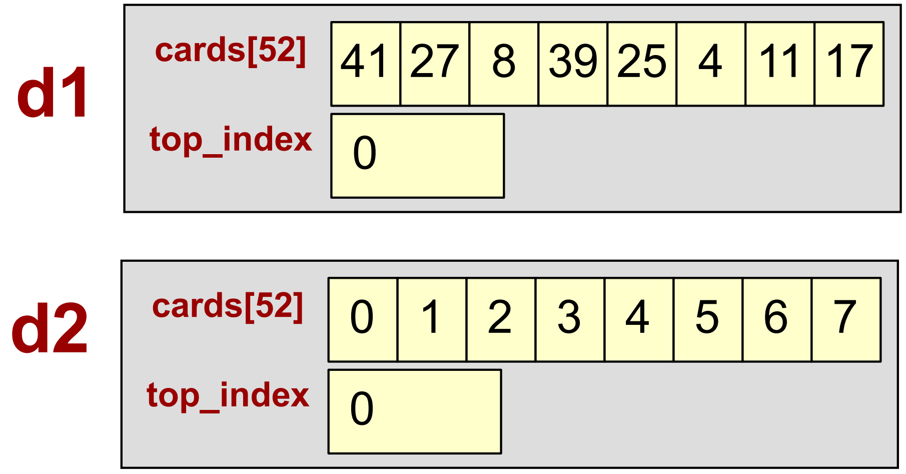
cards[52]	41	27	8	39	25	4	11	17
top_index	1							

Calling Member Functions

- Within a member function we can just call other member functions directly.

d1's data will be modified (shuffled and cut)

d1 is implicitly passed to shuffle()



d1

cards[52]	41	27	8	39	25	4	11	17
top_index	0							

d2

cards[52]	0	1	2	3	4	5	6	7
top_index	0							

Since shuffle was implicitly working on d1's data, d1 is again implicitly passed to cut()

poker.cpp

```
#include<iostream>
#include “deck.h”

int main(int argc, char *argv[]) {
  Deck d1, d2;
  int hand[5];
  d1.shuffle();
  ...
}
```

deck.cpp

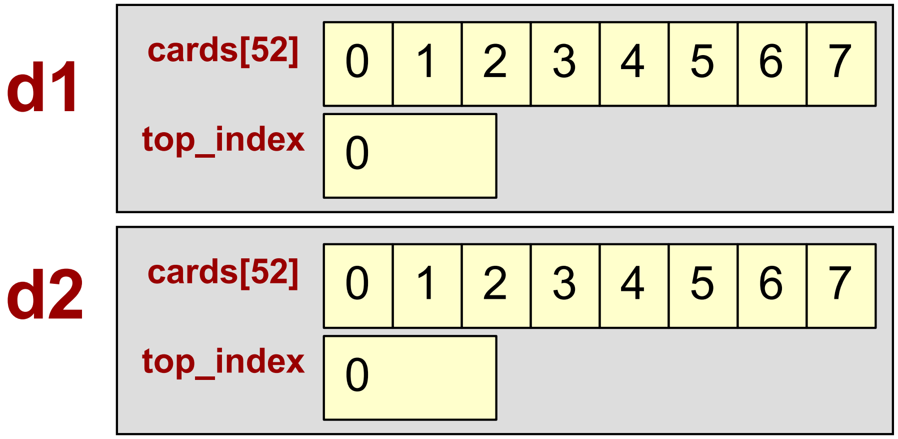
```
#include<iostream>
#include “deck.h”

void Deck::shuffle()
{
  cut(); // calls cut()
         // for this object
  for(i=0; i < 52; i++){
    int r = rand() % (52-i);
    int temp = cards[r];
    cards[r] = cards[i];
    cards[i] = temp;
  }
}
void Deck::cut()
{
  // swap 1st half of deck w/ 2nd
}
```

Exercises

- In-class Exercises

Class Pointers

- Can declare pointers to these new class types
- Use '->' operator to access member functions or data



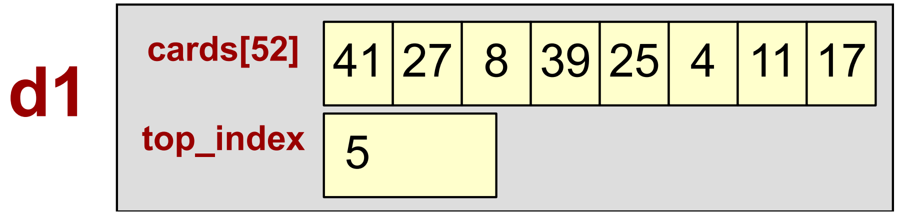
d1

cards[52]	0	1	2	3	4	5	6	7
top_index	0							

d2

cards[52]	0	1	2	3	4	5	6	7
top_index	0							

```
#include<iostream>
#include "deck.h"

int main(int argc, char *argv[]) {
  Deck *d1;
  int hand[5];
  d1 = new Deck;
  d1->shuffle();
  for(int i=0; i < 5; i++){
    hand[i] = d1->deal();
  }
}
```



d1

cards[52]	41	27	8	39	25	4	11	17
top_index	5							

Multiple Constructors

- Can have multiple constructors with different argument lists

```
#include<iostream>
#include “deck.h”

int main()
{
  Student s1;   // calls Constructor 1
  string myname;
  cin >> myname;
  s1.set_name(myname);
  s1.set_id(214952);
  s1.set_gpa(3.67);

  Student s2(myname, 32421, 4.0);
             // calls Constructor 2

}
```

deck.h

```
class Student {
 public:
   Student();    // Constructor 1
   Student(string name, int id, double gpa);
                 // Constructor 2
   ~Student();   // Destructor
   string get_name();
   int get_id();
   double get_gpa();

   void set_name(string name);
   void set_id(int id);
   void set_gpa(double gpa);
 private:
   string _name;
   int _id;
   double _gpa;
};
```

deck.cpp

```
Student::Student()
{
  _name = “”, _id = 0; _gpa = 2.0;
}

Student::Student(string name, int id, double
gpa)
{
  _name = name; _id = id; _gpa = gpa;
}
```

Public / Private and Structs vs. Classes

- In C++ the only difference between structs and classes is structs default to public access, classes default to private access
- Thus, other code (non-member functions of the class) ***cannot*** access private class members directly

student.h

```
class Student {
 public:
   Student();    // Constructor 1
   Student(string name, int id, double gpa);
                 // Constructor 2
   ~Student();   // Destructor
   ...
 private:
   string _name;
   int _id;
   double _gpa;
};
```

grades.cpp

```
#include<iostream>
#include “student.h”
int main()
{
  Student s1;  string myname;
  cin >> myname;
  s1._name = myname; //compile error
  ...
}
```

Accessor / Mutator Methods

- Define public “get” (accessor) and “set” (mutator) functions to let other code access desired private data members
- Use 'const' after argument list for accessor methods

```
#include<iostream>
#include “deck.h”

int main()
{
  Student s1;  string myname;
  cin >> myname;
  s1.set_name(myname);

  string another_name;
  another_name = s1.get_name();

  ...
}
```

student.h

```
class Student {
 public:
   Student();    // Constructor 1
   Student(string name, int id, double gpa);
                 // Constructor 2
   ~Student();   // Destructor
   string get_name() const;
   int get_id() const;
   double get_gpa() const;

   void set_name(string s);
   void set_id(int i);
   void set_gpa(double g);
 private:
   string _name;
   int _id;
   double _gpa;
};
```

student.cpp

```
string Student::get_name()
{   return _name; }
int Student::get_id()
{   return _id; }
void Student::set_name(string s)
{  _name = s;  }

void Student::set_gpa(double g)
{  _gpa = g; }
```